Docket No. 2000-0218

UNITED STATES PATENT APPLICATION

METHOD FOR STREAM MERGING

INVENTORS:

Amotz Bar-Noy

Richard Ladner

Cross Reference to Related Applications

This application claims priority to United States Provisional Application Serial No. 60/195,972, entitled "Off-Line And On-Line Stream Merging," filed on April 11, 2000, the contents of which is incorporated by reference herein.

METHOD FOR STREAM MERGING

Field of the Invention

The present invention relates generally to media streaming, and, more particularly, to optimizing multicast delivery of a media stream to a plurality of clients in a communication network.

Background of the Invention

The simplest policy for serving a media stream to a plurality of clients – e.g., in a video or audio-on-demand system – is to allocate a new media stream for each client request. Such a policy, however, is very expensive, as it requires server bandwidth that scales linearly with the number of clients. One of the most common techniques for reducing server bandwidth is to "batch" multicasted streams into scheduled intervals. Clients that arrive in an interval are satisfied by a full stream at the end of the interval. Bandwidth is saved at the expense of longer guaranteed startup delay.

One recent proposal to reduce server bandwidth is to use a server delivery policy referred to as "stream merging." See, e.g., D. L. Eager, M. K. Vernon, and J. Zahorjan, "Minimizing bandwidth requirements for on-demand data delivery," Proceedings of the 5th International Workshop on Advances in Multimedia Information Systems (MIS '99), 80-87, 1999. Stream merging assumes that there are multicast media channels and that each client has adequate buffer space and receive bandwidth that is at least twice the playback bandwidth. Under stream merging, the client receives two (or more) channels of the media stream: one channel starting from the beginning of the stream, a second channel commencing mid-stream, e.g. as it is being multicast to other clients who have arrived at an earlier time. The client commences processing of the first channel while buffering the second channel. When the first channel reaches the point in the stream corresponding to the beginning of the buffered stream from the second channel, the client switches to the buffered stream (thereby "merging" the streams) and the transmission on the first channel may be dropped – thereby saving bandwidth.

Summary of the Invention

The present invention is directed to a system and method for stream merging which improves upon the prior art by utilizing optimized merging patterns. In accordance with an embodiment of the present invention, the server, channels, and clients in the stream merging architecture have specific and well-defined roles. The server informs the client which streams to monitor and for how long; the server advantageously need only communicate with the client during setup of the media stream. In accordance with another embodiment of the present invention, the server optimizes the merging of multiple client streams by minimizing the cost of different merge patterns. Optimal solutions are disclosed for when stream initiations are both known and unknown ahead of time. Where streams initiations are regular and known ahead of time, optimal merging patterns can be calculated using a novel closed form solution for the merge cost. Where the stream initiations are not regular, the server can utilize the property of monotonicity to quickly calculate optimal merge patterns. Where stream initiations are not known ahead of time, the server can readily decide whether to initiate a new stream or whether to merge the new stream into the existing merge tree, advantageously into the right frontier of the merge tree. The inventors disclose that optimal merge trees have interesting relationships to Fibonacci number recurrences and that a Fibonacci merge tree structure can be advantageously used in an on-line stream merging system.

These and other advantages of the invention will be apparent to those of ordinary skill in the art by reference to the following detailed description and the accompanying drawings.

Brief Description of the Drawings

FIG. 1 illustrates a multicast network with a server and multiple clients.

FIG. 2 is a representation of the components of the server and clients in FIG. 1.

FIG. 3 is a timeline illustrating the process of stream merging.

FIG. 4 is a flowchart of processing performed by a client, in accordance with an embodiment of the invention.

FIG. 5 is a flowchart of processing performed by a server using off-line stream merging, in accordance with an embodiment of the invention.

FIG. 6 is a conceptual representation of a merge tree, corresponding to the stream merging example shown in FIG. 3.

FIG. 7 is an abstract diagram illustrating the recursive structure of a merge tree T with root r.

FIG. 8A through 8D are conceptual representations of Fibonacci merge trees for $n = 3, 5, 8, 13$, respectively. FIG. 8E and 8F illustrate two merge trees for $n = 4$.

FIG. 9 shows the values of $I(n)$ for $2 \le n \le 34$.

FIG. 10 is a flowchart of processing performed by a server using on-line stream merging, in accordance with an embodiment of the invention.

Fig. 11 is an abstract diagram illustrating the transformation from T_{n-1} to T_{n-1}^{i} in the basic merging rule in on-line stream merging.

Fig. 12 is an abstract diagram illustrating the transformation from T to T^x.

Fig. 13 is an abstract diagram illustrating the transformation from T_{n-1} to T_{n-1}^{i} in a dynamic tree algorithm.

Detailed Description

In FIG. 1, a plurality of media clients 110, 120, … 130 are provided access to media streams by a multicast-enabled network 100, as is well understood in the art. A media server 150 stores and multicasts particularly popular media on multiple channels 101 at different times across network 100 to satisfy client demands. Each client, e.g. client 110, issues a request to the server

150 for a media stream, otherwise referred to herein as an "arrival" at the server 150. At each arrival time, a stream is scheduled by the server 150, although for a given arrival the stream may not run until conclusion because only an initial segment of the stream is needed by the client 110. The server 150 issues a response to the client 101 that informs the client 101 which streams to monitor and for how long. The request and the corresponding response can be made using any known or arbitrary communication protocol. After this exchange, the client 101 needs no further interaction with the server 150. The client 110 receives and buffers data from two or more streams at the same time, in accordance with the response from the server 150, while a user can "play" or "view" the data accumulated in the client buffer. Each client 110...130 can receive all the parts of the media stream and play them without any interruption starting right after the time of its arrival.

FIG. 2 is a conceptual diagram of the components of the client 210 and server 250, corresponding to the client 110 and server 150 in FIG. 1. The server 250 comprises a computational engine 251, which constructs optimized stream merging patterns, as further described herein, connected to an external or internal storage device 252 which may be used for the storage of the media stream(s). The computational engine 251 controls a network interface 255 which forwards the relevant media streams at the relevant times through the multicast network 100. Although the exact number of channels is not relevant to the invention, four multicast channels 201, 202, 203, 204 are shown in FIG. 2. The client 210 has its own network interface 215 capable of receiving data from the multicast channels 201-204. The client 210 has its own computational engine 211, which merely follows the stream merging rules and the receiving procedure described below. The client's computational engine 211 directs and stores data received from particular multicast channels to a memory buffer 212. The client 210 can have a player component 213, which is capable of presenting the data in the media stream to a user. At the top of FIG. 2, the client 210 is depicted commencing the processing of a media stream, after obtaining a receiving procedure from the server 250. As shown in the bottom of FIG. 2, the client 210 buffers the data received from two multicast channels while simultaneously sending the initial parts of the stream to the player component 213. The exact nature of the processing performed by the client 210 and the server 250 is now described in further detail.

For purposes of describing the different embodiments of the invention, it is advantageous to use a discrete time model, as illustrated by the timeline shown in FIG. 3. The horizontal axis is the time axis and the vertical axis shows the particular unit of the full stream that is transmitted. Time is assumed to be slotted into unit sized intervals, each slot t starting at time $t - 1$ where the length of a full stream is L units. Let $t_1, t_2, \ldots, t_n$ be a sequence of arrival times for clients. Clients that arrive at the same time slot can be considered as one client and serviced in the same manner. At each arrival time, a new stream is initiated – although for a given arrival the stream may be truncated in the context of the stream merging process. The client arrival time is used herein interchangeably to identify the client(s) arrival and the stream initiated at the particular time. The time interval can be a reflection of the delay constraints of the media streaming system: e.g. a two hour streaming movie which can tolerate a 4 minute startup delay can be configured with a time interval of 4 minutes making each movie $L =$ 30 units long. Note that although the invention is presented in the context of a discrete time model, it is readily extendible to a non-discrete time model by letting the time slots be as small as desired and where the value of L is as large as needed. FIG. 3 shows a full stream of some length $L > 24$ commenced at time slot 1 with a series of other streams commenced at later times and truncated and merged into the full stream.

FIG. 4 is a flowchart of the processing performed by a media client 110, in accordance with a preferred embodiment of the invention. At step 401, the media client 110 issues a request for a media stream to the media server 150. As described in further detail below, the server 150 constructs a stream merging pattern and, at step 402, returns a schedule of arrival times for a plurality of $k + 1$ streams denoted as $x_0, x_1, \ldots, x_k$ and referred to herein as a receiving procedure for the client. Thereafter, the client 110 needs no further communication with the media server 150 and, at steps 403 to 408, can merely "listen" to the identified multicast channel at the particular associated time periods represented in the receiving procedure. At step 403, the counter i is set to 0. From time slot x_k until time slot $2x_k - x_{k-1}$, the client receives different parts of the requested media stream from two different multicast channels. At 405, the client receives the beginning of the requested stream, namely parts $1, \ldots, x_k - x_{k-1}$, from stream x_k and can immediately begin utilizing the stream. Simultaneously at 406, the client buffers the parts $x_k - x_{k-1} + 1, \ldots, 2x_k - 2x_{k-1}$ from stream x_{k-1}. At step 407, the counter i is incremented by 1 and the steps 404 to 406 are repeated until i equals $k - 1$. From time slot $2x_k - x_{k-i}$ until time slot $2x_k - x_{k-i-1}$, parts $2x_k - 2x_{k-i} + 1, \ldots, 2x_k - x_{k-i} - x_{k-i-1}$ are received from stream x_{k-i} while parts $2x_k - x_{k-i} - x_{k-i-1} + 1, \ldots, 2x_k - 2x_{k-i-1}$ are received from stream x_{k-i-1}. The parts are buffered and played as needed. Finally, at step 408, the last parts of the media stream $2(x_k - x_0) + 1, \ldots, L$ are received and buffered from stream x_0 from time slot $2x_k - x_0$ until time slot $x_0 + L$. Note that although FIG. 4 illustrates the invention with two multicast receiving streams, the invention is not limited to the "receive-two" model shown and described herein. One of ordinary skill in the art can readily extend the embodiment to multiple multicast receiving streams, although it can be shown that the benefits of adding receiving bandwidth become marginal.

The media client 110 advantageously avoids complex computations and need only follows the basic stream merging rules reflected in FIG. 4. As an example of the processing performed in FIG. 4, consider the stream merging pattern set forth in FIG. 3. A full stream of length L has already been commenced at time slot 1. The client, upon issuing a media stream request just before time slot 13, is issued a receiving procedure of streams $x_0 = 1$, $x_1 = 9$, $x_2 = 12$, $x_3 = 13$, where $k = 3$. At time slot 13 (where the counter $i = 0$ in FIG. 4), the client receives the first part of the stream from stream x_3 while buffering part 2 from stream x_2. For the next three time slots ($i = 1$), the client receives and buffers parts 3-5 from stream x_2 while receiving and buffering parts 6-8 from stream x_1. For the next eight time slots ($i = 2$), the client receives and buffers parts 9-16 from stream x_1 while receiving and buffering parts 17-24 from stream x_0. Finally at the last step in FIG. 4, the client receives the remaining parts of the stream from the full stream x_0.

The media server 150 is responsible for computing the stream merging patterns and for disseminating the proper receiving procedures to its clients. FIG. 5 sets forth a simplified flowchart of the processing performed by the server 150 in the "off-line" situation, i.e. where the stream initiations are known ahead of time. At step 501, the server 150 receives reservation requests in advance from each of the clients. At step 502, the server 150 calculates an optimal merging schedule with corresponding receiving procedures $x_0, \ldots, x_k$ for each client, and, at step 503, the server 150 transmits the receiving procedures to each client. At step 504, the server 150 commences the multicast transmissions, in accordance with the schedule calculated at step 502.

A preferred method of calculating the merging schedule would be to optimize the "cost" of different merging patterns. For example, the server could minimize the sum of the lengths of all of the streams in the merging pattern, which would be equivalent to minimizing the total number of units (total bandwidth) needed to serve all the clients. In that context, and in accordance with an aspect of the invention, FIG. 6 illustrates a particularly helpful abstraction of the diagram set forth in FIG. 3. The inventors refer to the abstraction as a "merge tree." A merge tree is an ordered labeled tree, where each node 601-608 is labeled with an arrival time and the stream initiated at that time. For example, nodes 601, 602, 603 and 604 correspond to the arrival times/streams $x_0 = 1$, $x_1 = 9$, $x_2 = 12$, $x_3 = 13$, described above. Each new stream can only merge to an earlier stream, and the children of a given node are ordered by their arrival times. If a preordered traversal of the labeled tree yields the arrival times in order, as does the tree illustrated in FIG. 6, the tree is said to have a "preorder traversal property." An optimized solution for a given client arrival sequence is a merging pattern which can be represented as a sequence of merge trees, which the inventors refer to as a "merge forest."

Given a merge tree T, the root of the tree represents a full stream of length L and is denoted by $r(T)$. If x is a node in the merge tree, $\ell(x)$ is defined as the length in T; that is, $\ell(x)$ is the minimum length needed to guarantee that all the clients can receive their data from stream x using the stream merging rules. A helpful distinction can be made between "merge cost" and "full cost" where the merge cost includes just the cost of merging and not the full stream which is the target of the merging. The merge cost is defined as

$$\mathrm{Mcost}(T) = \sum_{x \neq r(T) \in T} \ell(x)$$

That is, the merge cost of a tree is the sum of all lengths in the tree except the length of the root of the tree. The full cost counts everything: merging cost and full stream cost for all of the merge trees in the forest. The optimal merge cost is defined as the minimum cost of any merge tree for the sequence. An optimal merge tree is one that has optimal merge cost. There is a simple formula for calculating the minimum length required for each node of a merge tree. Let $x \neq r(T)$ be a non-root node in a tree T. Then

$$\begin{aligned}\ell(x) &= 2z(x) - x - p(x)\\ &= (z(x) - x) + (z(x) - p(x))\end{aligned}$$

where $z(x)$ is the arrival time of the last stream in the subtree rooted at x and $p(x)$ is a parent of x. In particular, if x is a leaf, then $z(x) = x$ and $\ell(x) = x - p(x)$. The length of stream x is composed of two components: the first component is the time needed for clients arriving at time x to receive data from stream x before they can merge with stream $p(x)$; the second component is the time stream x must spend until the clients arriving at time $z(x)$ merge to $p(x)$. Using the preorder traversal property of optimal merge trees, there is an elegant recursive formula for the merge cost of a tree T, illustrated by Fig. 7. A key property of merge trees is that for any node t_i, the subtree rooted at t_i contains the interval of arrivals t_i, $t_{i+1}, \ldots, t_j$, where $z(t_i) = t_j$. Furthermore, t_j is the right most descendant of t_i. As a result, any merge tree can be recursively decomposed into two in a natural way as illustrated in FIG. 7. FIG. 7 shows the recursive structure of a merge tree T with root *r*. The last arrival to merge directly with *r* is *x*. All the arrivals before *x* are in *T'* and all the arrivals after *x* are in *T''* and *z* is the last arrival. Thus, it can be shown that

$$\text{Mcost}(T) = \text{Mcost}(T') + \text{Mcost}(T'') + 2z - x - r$$

The full cost of a forest *F* of *s* merge trees $T_1, \ldots, T_s$, is defined as

$$\text{Fcost}(F) = s \cdot L + \sum_{1 \le i \le s} \text{Mcost}(T_i)$$

that is, the full cost of a forest is the sum of the merge costs of all its trees plus *s* times the length of a full transmission, one per each tree. Note that the length of any non-root nodes in *T* cannot be greater than *L*. Merge trees that do not violate this condition are referred to by the inventors as "L-trees." The optimal full cost for a sequence is the minimum full cost of any such forest for the sequence. An optimal forest is referred to as one that has optimal full cost.

Define $M(i, j)$ to be the optimal merge cost for the input sequence $t_i, \ldots, t_j$. The optimal cost for the entire sequence, thus, is $M(1,n)$. The optimal cost may be computed using dynamic programming. $M(i, j)$ can be recursively defined as follows

$$M(i, j) = \min_{i<k\le j}\{M(i,k-1) + M(k, j) + (2t_j - t_k - t_i)\}$$

with the initialization $M(i,i) = 0$. This recursive formulation naturally leads to an $O(n^3)$ time algorithm using dynamic programming, as is well understood in the art. The time to compute the optimal merge cost may be reduced to $O(n^2)$ be employing the classic technique of monotonicity. See, e.g., D. E. Knuth, "Optimum Binary Search Trees," Acta Informatica, Vol. 1, 14-25 (1971). Define $r(i,i) = i$ and, for $i < j$, as follows:

$$r(i, j) = \max\{k : M(i, j) = M(i,k-1) + M(k, j) + 2t_j - t_k - t_i\}$$

Thus, $r(i, j)$ is the last arrival that can merge to the root in some optimal merge tree for $t_i,...,t_j$. Monotonicity is the property that for $1 \le i < n$ and $1 < j \le n$

$$r(i, j-1) \le r(i, j) \le r(i+1, j) \quad .$$

It should be noted that there is nothing special about using the max in the definition of $r(i, j)$; the min would yield the same inequality. Monotonicity can be demonstrated using a very elegant method of quadrangle inequalities. See F. F. Yao, "Efficient Dynamic Programming Using Quadrangle Inequalities," Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC '80), 429-35 (1980); A. Borchers and P. Gupta, "Extending the Quadrangle Inequality to Speed Up Dynamic Programming," Information Processing Letters, Vol. 49, 287-90 (1994). Thus, the search for the *k* in the above recursive formulation can be reduced to $r(i+1, j) - r(i, j-1) + 1$ possibilities from $j-1$ possibilities. The key point is that the right most stream that merges to the root of an optimal tree from *i* to *i + j* is confined to an interval and these intervals are almost disjoint for *i* not equal to *j*.

An optimal algorithm for calculating the full cost uses the optimal algorithm for merge cost above as a subroutine. Let $t_1, t_2, ..., t_n$ be a sequence of arrivals, and let *L* be the length of a full stream. For $1 \le i \le n$, define $G(i)$ to be the optimal full cost for the last $n-i+1$ arrivals $t_i,...,t_n$. Define $G(n+1) = 0$ and for $1 \le i \le n$,

$$G(i) = L + \min\{M(i, k-1) + G(k) : i < k \le n+1 \text{ and } t_{k-1} - t_i \le L-1\} \quad .$$

The optimal full cost is $G(1)$ and the order of computation is $G(n+1), G(n), ..., G(1)$. The optimal full cost algorithm proceeds in two phases. In the first phase, the optimal merge cost $M(i, j)$ is computed for all *i* and *j* such that $0 \le t_j - t_i \le L-1$, so that these values can be used to compute $G(i)$. In the second phase, $G(i)$ is computed from $i = n$ down to 1 using the above equation. The intuition for the above is as follows: a full stream must begin at t_1, and there are two possible cases in an optimal solution. Either all the remaining streams merge to this first stream or there is a next full stream t_k for some $k \leq n$. In the former case, the optimal full cost is simply $L + M(1,n)$. In the latter case, the optimal full cost is $L + M(1,k-1)$ plus the optimal full cost of the remaining arrivals $t_k,...,t_n$. In both cases, the last arrival to merge to the first stream must be within $L-1$ of the first stream. The full streams can be identified inductively. Both phases of the optimal algorithm together run in time $O(nm)$, where m is the average number of arrivals in an interval of length $L-1$ that begins with an arrival. The above algorithm is practical enough to schedule millions of reserved arrivals.

An important special case which simplifies the above optimal merge cost solution is when an arrival is scheduled at every time unit, referred to herein as the "fully loaded arrivals" case. The fully loaded arrivals case can be thought of as being a system with a guaranteed maximum delay, where streams are scheduled at every time unit regardless of client arrivals. For the case of fully loaded arrivals, the value $M(i,j)$ does not depend on $i(t_i)$ and $j(t_j)$ but rather depends on their difference $j - i$. Hence, where $M(n)$ is the minimum cost for a merge tree for the arrivals $[0, n - 1]$, the following recursive formula for the merge cost for fully loaded arrivals is obtained.

$$M(n) = \min_{1 \leq h \leq n-1} \{M(h) + M(n-h) + 2n - h - 2\}$$

with the initialization $M(1) = 0$. Using the notation above, the term $2n - h - 2$ comes from $z = n - 1$, $x = h$, and $r = 0$ and then $2z - x - r = 2n - h - 2$. Calculating $M(n)$ for small values of n yields an interesting sequence:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$M(n)$	0	1	3	6	9	13	17	21	26	31	36	41	46	52	58	64

A careful examination of this sequence reveals that there is a very elegant formulation of the merge cost in terms of Fibonacci numbers:

$$M(n) = (k-1)n - F_{k+2} + 2 \qquad \text{if} \quad F_k \leq n \leq F_{k+1}$$

where F_k is the kth Fibonacci number. As is well known in the art, the Fibonacci numbers are defined by the following recurrence: $F_k = F_{k-1} + F_{k-2}$ for $k \geq 2$, where $F_0 = 0$ and $F_1 = 1$. It can be shown that for n equal to a Fibonacci number there is a unique optimal tree, which the inventors refer to as a "Fibonacci merge tree." FIG. 8A through 8D illustrate such optimal trees for $n = 3, 5, 8, 13$, with corresponding merge costs of $M(n) = 3, 9, 21, 46$, respectively. Note the structure of these optimal trees: the right-most subtree of the tree for $n = F_k$ is the tree for $n = F_{k-2}$ whereas the rest of the tree to the left is a tree for $n = F_{k-1}$. On the other hand, for other values of n there can be multiple optimal trees, e.g., FIG. 8E and FIG. 8F illustrate two optimal trees for four arrivals, both trees having a merge cost of six. It is of interest to see which arrivals can be the last to merge in an optimal merge tree. Define the following two auxiliary functions:

$$H(n,h) = M(h) + M(n-h) + 2n - h - 2$$
$$I(n) = \{h : M(n) = H(n,h)\}$$

so that the value of $M(n)$ can be determined by minimizing $H(n, h)$ for $1 \leq h \leq n - 1$. The members of $I(n)$ are all the arrivals that can be the last merge to the root in an optimal merge tree for $[0, n-1]$. FIG. 9 shows the values of $I(n)$ for $2 \leq n \leq 34$. Each set $I(n)$ is an interval and the pattern depends heavily on Fibonacci numbers. The following definitions are useful in characterizing these intervals. For a given n, $n = F_k + m$ for some $0 \leq m \leq F_{k-1}$, define the following three intervals:

$$I_1(n) = [F_{k-1}, F_{k-1} + m]$$
$$I_2(n) = [F_{k-2} + m, F_{k-1} + m]$$
$$I_3(n) = [F_{k-2} + m, F_k]$$

A given interval $I_i(n)$ will be the $I(n)$ for a certain range of m in the interval $[0, F_{k-1}]$. Define those ranges as:

$$m_1(k) = [0, F_{k-3}]$$
$$m_2(k) = [F_{k-3}, F_{k-2}]$$
$$m_3(k) = [F_{k-2}, F_{k-1}]$$

Then it can be shown by induction that if $m \in m_i(k)$,

$$M(n) = (k-1)n - F_{k+2} + 2 \quad \text{and} \quad I(n) = I_i(n)$$

which can be used as the basis of an efficient algorithm to construct an optimal merge tree for fully loaded arrivals.

An optimal merge tree for fully loaded arrivals can thus be computed in time O(n) using the above closed form solution. Let [0, n – 1] be an input. Define $r(i) = \max I(i)$ for $1 \le i \le n$. So $r(i)$ is an arrival that can be the last merge in an optimal merge tree for the input [0, i – 1]. An optimal merge tree for the input [i, j] can be computed using the following recursive procedure. If i = j return the tree with label i. Otherwise, recursively compute the merge tree T_1 for the input $[i, i + r(j-i+1) - 1]$ and T_2 for $[i + r(j-i+1), j]$, then attach the root of T_2 as an additional last child of the root of T_1 and return the resulting tree. This procedure is called for the input [0, n – 1] to construct an optimal merge tree. With an elementary data structure the tree can be constructed in linear time provided that $r(i)$ has already been computed for $1 \le i \le n$. The Fibonacci numbers $\le n$, can be computed in O(log n) time. The sequence $r(1)$, $r(2)$, …, $r(n)$ can be computed in linear time using the recurrence

$$\begin{aligned} r(i) &= r(i-1) + 1 \quad && \text{if} \quad F_k < i \le F_k + F_{k-2} \\ &= r(i-1) && \text{if} \quad F_k + F_{k-2} < i \le F_{k+1} \end{aligned}$$

with the initialization $r(1) = 0$ and $r(2) = 1$. An optimal forest for fully loaded streams can be constructed in linear time. In computing the full cost of a merge forest, the cost of the roots must be taken into account. There are basically two steps: first, determine how many full streams are in an optimal merge forest and, second, where to place the full streams. Define $F(L, n, s)$ to be the minimum cost of any merge forest for [0, n – 1] where the length of a full stream is L and there are exactly s roots (full streams). Since at most L – 1 streams can merge with a stream of length L, it follows that for a given n there must be at least $s_0 = \lceil n / L \rceil$ full streams for n arrivals. Hence,

$$F(L, n) = \min_{s_0 \le s \le n} F(L, n, s)$$

Notice the extreme cases: $L = 1$ implies $s_0 = n$ and $n = L - 1$ implies $s_0 = 1$.

For a fixed s, the placement of the full streams in an optimal merge forest with s full streams can be determined. Where $n = ps + r$ and $0 \le r < s$, it can be shown that

$$F(L,n,s) = sL + rM(p+1) + (s-r)M(p)$$

This yields a straightforward linear time algorithm for computing an optimal merge forest. First, the above-described Fibonacci formulation of M can be used to compute $M(1), M(2),\ldots, M(L)$. Next, search for an s ($s_0 \le s \le n$) that minimizes $sL + rM(p+1) + (s-r)M(p)$ where $p = \lfloor n/s \rfloor$ and $r = n - ps$. To construct the merge forest, place r full streams at $0, p+1, 2(p+1), \ldots, (r-1)(p+1)$ and $s-r$ full streams at $r(p+1), r(p+1)+p, r(p+1)+2p, \ldots, r(p+1)+(s-r-1)p$. Use the linear time algorithm for constructing an optimal merge tree to complete the forest. The optimal merge forest for fully loaded arrivals can be computed in $O(L + n)$ time. Note that it is possible to directly calculate the number of full streams needed in an optimal merge forest rather than searching for the s that minimizes the above expression. First, compute h such that $F_{h+1} < L + 2 \le F_{h+2}$. This h can be computed in linear number of log operations. Next, compute $s_1 = \lfloor n / F_h \rfloor$ and $s_0 = \lceil n/L \rceil$. If $s_0 > s_1$ then $s_0 = s_1 + 1$ minimizes $F(L, n, s)$. Otherwise, compute $F(L, n, s_1)$ and $F(L, n, s_1+1)$ using the above expression. If the former value is smaller, then s_1 minimizes $F(L, n, s)$, otherwise $s_1 + 1$ does. It is interesting to note that there are cases where s_1 is optimal and $s_1 + 1$ is not, $s_1 + 1$ is optimal and s_1 is not, and both s_1 and $s_1 + 1$ are optimal. It should be noted that a natural guess for s is $\lceil n/(\lfloor L/2 \rfloor + 1) \rceil$. That is, a full stream is scheduled at intervals of length about $L/2$ and each tree contains about $L/2$ nodes. This value of s is not always optimal, but it is optimal in many cases and at the very least gives a good upper bound for the full cost in the fully loaded arrivals case.

In contrast to the off-line situation in which client reservations are accepted in advance, we next describe the "on-line" situation in which the client requests are not known ahead of time. When a new client t_n arrives, the media server 150 is assumed to have already constructed a merge forest F_{n-1} for the preceding $n-1$ clients, $t_1, \ldots, t_{n-1}$ where t_1 is the root of the first merge tree in the forest. Given $n > 1$, a decision must be made to either incorporate t_n into the last merge tree in the forest or to start a new merge tree by making t_n its root. The goal in the on-line situation is to obtain results dynamically that are good relative to an after-the-fact off-line computation.

FIG. 10 sets forth a simplified flowchart of the processing performed by the server 150 in the on-line situation, in accordance with a preferred embodiment of the invention. At step 1001, the server 150 receives the request from the client at time slot t_n. At step 1002, the server 150 compares $t_n - t_m$ to the quantity $L/2$, where t_m is the last root of a merge tree in the merge forest F_{n-1}. If $t_n - t_m > L/2$, then, at step 1004, the server 150 starts a new merge tree with t_n becoming the root of the new merge tree in F_n. This start rule has many justifications. First, if $t_n - t_m \leq L/2$ then t_n can always be incorporated into the merge tree rooted at t_m, and for $t_n - t_m$ time slots the clients served by t_n will be receiving two streams simultaneously. Second, there is a serious disadvantage of trying to incorporate t_n into merge tree rooted at t_m if $t_n - t_m < L/2$. Consider the extreme example where $t_m = t_n - 1$ and $t_n = t_n - 1 + L - 1$. In this example, t_n can merge directly to t_{n-1} so that $\ell(t_n) = L-1$. However, stream t_n only receives one part of t_{n-1}, namely, its last part. Even worse, no future arrival can receive any part of t_n because doing so would cause the length of stream t_n to exceed L. The only potential gain in merging t_n to t_{n-1} is if there are no arrivals in the next L slots after t_n.

Assuming no new merge tree will be created, the server 150 must then decide how to incorporate the new arrival into the existing merge tree at step 1003. A new merge tree T_n will then be created which incorporates the arrival t_n into the existing merge tree, referred to as T_{n-1} for arrivals $t_1, \ldots, t_{n-1}$. In order to preserve the preorder labeling (and thereby the existing stream merging rules), the new arrival should be merged into the "right frontier" of the existing merge tree. The right frontier of T_{n-1} is the path $t_1 = x_0, x_1, \ldots, x_k = t_{n-1}$ where x_{i+1} is the right most child of x_i for $0 \leq i < k$. For example, consider the case of a new arrival at time slot 15 in FIG. 3. As reflected on the corresponding merge tree in FIG. 6, the right frontier of the merge tree comprises the nodes 601-604. The pre-order traversal property of the tree requires that some node on the right frontier be made the parent of the new arrival at slot 15. Note, however, that not every node on the right frontier is eligible to be a parent of the new arrival. The arrival at time slot 15 cannot merge with node 604 because the stream x_3 has already terminated. Thus, the arrival can be merged into the root at 601 (stream x_0) or into the remaining nodes 602 (stream x_1) or 603 (stream x_2). This basic merging rule can be expressed more formally as requiring that $T_n = T_{n-1}^0$ or $T_n = T_{n-1}^i$ for some $i > 0$ such that $t_n \le 2t_{n-1} - x_{i-1}$, where T_{n-1}^i is defined to be the tree T_{n-1} with x_i chosen as the parent of t_n, that is, $p_{T_{n-1}^i}(t_n) = x_i$. This transition from T_{n-1} to T_{n-1}^i is further illustrated abstractly by FIG. 11. Note that the new right frontier of T_{n-1}^i is $t_1 = x_0, x_1, \ldots, x_i, t_n$.

The incremental cost of merging t_n into T_{n-1} can be expressed as:

$$MCost(T_n) - MCost(T_{n-1}) = 2i(t_n - t_{n-1}) + t_n - x_i$$

where the last part of the cost, $t_n - x_i$, is the length of t_n and the first part of the cost represents the length of each non-root ancestor of t_n due to the change of its last descendant from t_{n-1} to t_n. A simple approach to optimizing on-line streaming would be to choose a parent so as to minimize the incremental merge cost, which the inventors refer to as a "best fit" rule. Another approach would be to pick a parent which is "closest" in some sense to the new arrival, which the inventors refer to as the "nearest fit" rule. For example, the largest i could be chosen where the i-th parent has not yet terminated. Unfortunately, it can be shown that these approaches do not have good performance relative to off-line stream merging.

Instead, and in accordance with another aspect of the invention, it can be shown that it is advantageous to force the on-line algorithm to "follow" an on-line merge tree as closely as possible. The on-line tree acts as a kind of "governor" in a tree-fit algorithm where each new arrival must merge with a member of the right frontier of the on-line merge tree. First, consider the situation of a fixed merging pattern, where the sequence of arrivals is not known in advance, but its length is assumed to be *n*. A fixed unlabeled tree *T* with *n* nodes, referred to by the inventors as a "static" merge tree, is utilized in an "oblivious" off-line merging process, which can be considered a "semi" on-line algorithm. Given an arrival sequence $\tau = (t_1, \ldots, t_n)$, a merge tree $T(\tau)$ is constructed with the same structure as *T*, but with the labels $t_1, \ldots, t_n$ put on the nodes in a preorder fashion. Hence, given two arrival sequences $\tau \neq \tau'$, it could be the case that Mcost($T(\tau)$) $\neq$ Mcost($T(\tau')$). How well a static merge tree *T* performs can be expressed as an approximation ratio a_T defined as follows:

$$a_T = \sup\{\frac{MCost(T(\tau))}{M_{opt}(\tau)} : \tau \text{ is an arrival sequence of length } n\ \}$$

This quantity measures the worst case performance of the static tree *T* as compared with optimal. It turns out that the approximation ratio of a static merge tree can be exactly characterized by measuring what the inventors refer to as its "extent." For a static merge tree *T* and a node *x* in *T*, define $u_T(x)$ to be the number of ancestors of *x* not counting *x* and the root of *T*, and define $v_T(x)$ to be the number of right siblings of ancestors of *x* (including *x*). The extent of x is defined to be:

$$e_T(x) = 2u_T(x) + v_T(x) + 1$$

while the extent of the static merge tree *T* is:

$$e(T) = \max\{e_T(x) : x \text{ in } T\}$$

For any static merge tree *T*, it can be shown that $a_T = e(T)$. The extent can be shown to be a lower bound of the approximation ratio by example. The extent can also be shown to be an upper bound by induction on the number of nodes in T using a transformation of T to an optimal T^x. The goal of the transformation is to make x, a node in T which is not the root, the last child of the root of T^x. The tree T^x is formed from two trees T_L^x and T_R^x as follows. First, T_L^x is the subtree of T consisting of all nodes that come before x in a preorder traversal of T. What remains from T after T_L^x is a sequence of disconnected merge trees $X_0, X_1, \ldots, X_k$ where X_0 is the subtree of T rooted at x and X_{i+1} is the subtree of T that is traversed in a preorder traversal of T immediately after X_i is traversed. The merge tree T_R^x is formed by taking X_0 whose root is x and making x the parent of the root of each X_i for $1 \leq i \leq k$ in that order. The transformation from T to T^x is illustrated in FIG. 12. Note that T_L^x is the subtree of T^x of all arrivals before x and T_R^x is the subtree of T^x of all arrivals after and including x. This corresponds to the subtrees T' and T'', respectively shown in FIG. 7. As a byproduct of the construction the subtree to the left of the last merge and the subtree rooted at the last merge each have extent less or equal to the extent of T. If the costs of the move are carefully examined, it can be shown that the cost is bounded by $e(T) - 1$ times the cost of x in T^x. The extent, and accordingly the approximation ratio for any static merge tree, can be shown to have a lower bound of $\log_\phi n - 1$, where $\phi = (1+\sqrt{5})/2 \approx 1.618$ is the golden ratio and is the positive root of the equation $x^2 = x + 1$. Furthermore, it is advantageous to note that the extent of a Fibonacci tree is essentially the same as the lower bound.

Given the knowledge of the approximation ratio for static trees, a new class of dynamic tree algorithms can be defined. Define a "preorder tree" to be an infinite tree in which the root has an infinite number of finite size subtrees as its children. Such a tree has the property that the preorder traversal provides a numbering for the entire tree starting with the root numbered 1. Define $T[n]$ to be the finite subtree of T of the nodes numbered 1 to n. An "infinite merge tree" is a preorder tree labeled with the arrival sequence $t_1, t_2, \ldots$ in preorder fashion. An advantageous example of a preorder tree what the inventors refer to as the infinite Fibonacci tree $\mathcal{F}$. Define the finite Fibonacci trees FT_1, FT_2, ... as follows. The trees FT_1 and FT_2 are each signal nodes. The tree FT_k is formed from FT_{k-1} and FT_{k-2} by making the root of FT_{k-2} the last child of the root of FT_{k-1}. It should be clear from the construction of FT_k that its size is F_k. Furthermore, it can be shown by induction that the extent $e(FT_k) = k - 2$ for $k > 2$. Define $\mathcal{F}$ as a root with infinitely many children where the kth child is the root of the subtree FT_k. A preorder traversal of $\mathcal{F}$ defines the preorder numbering of the nodes with the root numbered 1. Define $\mathcal{F}[n]$ to be the subtree of $\mathcal{F}$ consisting of the n nodes numbered from 1 to n. Then, it can be shown for $k \geq 2$, $\mathcal{F}[F_k] = FT_k$. The infinite Fibonacci tree yields static trees with almost minimal approximation ratio. For $n > 1$, it can be shown that $a_{\mathcal{F}[n]} \leq \lfloor \log_\phi n \rfloor$. Thus, the approximation ratio of $\mathcal{F}[n]$ is within 1 of the lower bound for all static trees of size n. If n is a Fibonacci number then $\mathcal{F}[n]$ has the minimum approximation ratio for a static tree of its size.

In accordance with an embodiment of an aspect of the invention, a dynamic tree algorithm proceeds by producing a new infinite merge tree for each new arrival. Suppose that T_{n-1} is the infinite merge tree after processing the arrivals $t_1, \ldots, t_2$,. Let $t_1 = y_0, y_1, \ldots, y_{k+1} = t_n$ be the path from the root to t_n in T_{n-1}. For each $i \leq k$, we define T_{n-1}^i which is formed from T_{n-1} as follows. Let $C_i = \{x : p(x) = y_i \text{ and } x > t_n\}$. So $x \in C_i$ if x is a child of y_i arriving later than t_n. Define T_{n-1}^i to be the tree T_{n-1} modified so that $p_{T_{n-1}^i}(t_n) = y_i$ and $p_{T_{n-1}^i}(x) = t_n$ for all $j > i$ and $x \in C_j$. See FIG. 13 for an illustration of the transformation T_{n-1} to T_{n-1}^i. The dynamic tree algorithm for T satisfies the following formal rule: either $T_n = T_{n-1}^0$ or $T_n = T_{n-1}^i$ for some i such that $0 < i \leq k$ and $t_n \leq 2t_{n-1} - y_{i-1}$. This is a special case of the basic merging rule described above. The path $y_0, \ldots, y_k$ is a prefix of the right frontier, which is the path from y_0 to t_{n-1}. It should be noted that although T_{n-1} is fully labeled with arrivals (suggesting that it is necessary to know the future arrivals and maintain an infinite tree), it can be assumed for implementation purposes that it is only labeled with the known arrivals $t_1, \ldots, t_{n-1}$. The algorithm knows the structure of the tree T_{n-1} so that when t_n becomes known it is made the label of the nth node in the tree. It can be seen inductively that T_n is an infinite preorder tree if T_{n-1} is an infinite preorder tree. The tree T is, by definition, composed of infinitely many finite trees T_1, T_2, ..., whose root is a child of the root of T. The tree T_{i-1} is numbered before the tree T_i in a preorder traversal of T. Let n_i be 1 plus the sum of the sizes of T_j for $j > i$. As long as $n \leq n_i$, only that part of T that includes the first i trees need be maintained. When $n = n_i + 1$, the next tree T_{i+1} can be incorporated into the algorithm. This can be done because if $n \leq n_i$, the transition from T_n to T_{n-1}^i leaves fixed all the trees T_j for $j > i$.

It would be advantageous that the new tree, in the transition from T_n to T_{n-1}^i, be just as effective as the old tree for future arrivals in order for the dynamic tree algorithm to behave well. This turns out to be true if the algorithm is "cost-preserving," meaning that the new tree $T_n = T_{n-1}^i$ for some $0 < i \leq k$ such that $y_i - y_k + 2(k-i)(t_n - t_{n-1}) \geq 0$. It can be shown that if this condition is true, then Mcost($T_{n-1}[m]$) $\geq$ Mcost($T_{n-1}^i[m]$) for all $m \geq n$. It can then be shown that if A is a dynamic tree algorithm for T that satisfies the cost preserving rule, then for all n, $c_A(n) \leq a_{T[n]}$. Thus, the competitive ratio performance of the dynamic tree algorithm can be related with the approximation ratio of the static trees. Two classes of algorithms can be easily shown to satisfy the cost preserving rule, and therefore are bounded above by the approximation ratios of the prefixes of T, $a_{T[n]}$: the "best fit" dynamic tree algorithm and the "nearest fit" dynamic tree algorithm for T. The "best fit" algorithm satisfies the rule that $T_n = T_{n-1}^i$ for an i that minimizes $\Delta_{i,n} = 2i(t_n - t_{n-1}) + t_n - y_i$. The "nearest fit" algorithm satisfies the rule that $T_n = T_{n-1}^0$ if $t_n > 2t_{n-1} - y_{i-1}$ for all $0 < i \le k$ and $T_n = T_{n-1}^i$ if i is the largest number such that $t_n \le 2t_{n-1} - y_{i-1}$.

Since the infinite Fibonacci tree F has the best approximation ratios in the static situation, it makes sense to use it in a dynamic tree algorithm. Where the best fit dynamic tree algorithm uses an infinite Fibonacci tree (referred to by the inventors as a "best fit dynamic Fibonacci tree (BFDT) algorithm") and where the nearest fit dynamic tree algorithm uses an infinite Fibonacci tree (referred to by the inventors as a "nearest fit Fibonacci tree (NFDT) algorithm"), it can be shown that the merge cost competitive ratios are bounded by $\lfloor \log_\phi n \rfloor$. For the case in which there is an arrival every time slot, both algorithms have a constant competitive ratio. They are optimal when n is a Fibonacci number, since the Fibonacci tree is the optimal merge tree for such sequences, as discussed above. This is not the case with other values of $n = F_k + m$, for $k \ge 3$ and $0 < m < F_{k-1}$. In this case, the optimal tree divides the arrivals into the left and the right subtrees according to the golden ratio. On the other hand, the size of the left subtree is always F_k. Nevertheless, the loss is not too big, and the competitive ratio is constant. Moreover, it can be shown that there is a bound on the full cost competitive ratios of the algorithms. Expressed using the parameter $D = 1/L$ (which can be interpreted as the guaranteed maximum startup delay measured as a percentage of the stream length), the full cost competitive ratios of the best and nearest fit dynamic Fibonacci tree algorithms are bounded above by:

$$\min\{\lfloor \log_\phi(1/(2D)) \rfloor + 2, \lfloor \log_\phi(n) \rfloor + 2\}$$

When D is very small the competitive ratio in the full cost is $O(\log n)$ as is the competitive ratio for the merge cost. In the extreme, when D tends to zero, this models situations in which arrivals could happen at any time. However, it is very realistic to assume that n is very large and D is a constant. That is, clients tolerate some delay and the time horizon is long. In this case, the above equation yields a constant competitive ratio bound. As an example, suppose there is a two hour video with a guaranteed maximum delay of 4 minutes. Then $L = 30$ and $D = 1/30$ or about 3.33%. The best fit and nearest fit dynamic Fibonacci tree algorithms have competitive ratios bounded above, according to the above equation, by 8. Hence, it is known that these algorithms will never use more than 8 times the bandwidth required by an optimal off-line solution—and in common case arrivals will perform even better.

The foregoing Detailed Description is to be understood as being in every respect illustrative and exemplary, but not restrictive, and the scope of the invention disclosed herein is not to be determined from the Detailed Description, but rather from the claims as interpreted according to the full breadth permitted by the patent laws. It is to be understood that the embodiments shown and described herein are only illustrative of the principles of the present invention and that various modifications may be implemented by those skilled in the art without departing from the scope and spirit of the invention. For example, many of the examples and equations have been presented in the context of a model in which the client receives data from two multicast channels. One of ordinary skill in the art can readily extend the various aspects of the above invention to clients that receive data from more than two multicast channels.